Factors Impacting the Effort Required to Fix Security Vulnerabilities

An Industrial Case Study

Lotfi ben Othmane[1], Golriz Chehrazi[1], Eric Bodden[1], Petar Tsalovski[3],
Achim D. Brucker[3], and Philip Miseldine[3]

[1] Fraunhofer Institute for Secure Information Technology
Darmstadt, Germany
{lotfi.ben.othmane,golriz.chehrazi,eric.bodden}@sit.fraunhofer.de

[2] SAP SE
Walldorf, Germany
{petar.tsalovski,achim.brucker,philip.miseldine}@sap.com

Abstract. To what extent do investments in secure software engineering pay off? Right now, many development companies are trying to answer this important question. A change to a secure development lifecycle can pay off if it decreases significantly the time, and therefore the cost required to find, fix and address security vulnerabilities. But what are the factors involved and what influence do they have? This paper reports about a qualitative study conducted at SAP to identify the factors that impact the vulnerability fix time. The study involves interviews with 12 security experts. Through these interviews, we identified 65 factors that fall into classes which include, beside the vulnerabilities characteristics, the structure of the software involved, the diversity of the used technologies, the smoothness of the communication and collaboration, the availability and quality of information and documentation, the expertise and knowledge of developers, and the quality of the code analysis tools. These results will be an input to a planned quantitative study to evaluate and predict how changes to the secure software development lifecycle will likely impact the effort to fix security vulnerabilities.

Keywords: Human factors, secure software, vulnerability fix time

1 Introduction

Despite heavy investments into software security [1], security experts and attackers continue to discover code vulnerabilities in software systems on a regular basis, including buffer overflows, SQL injections, and unauthorized procedure calls. While some attack vectors relate to mis-designed software architectures, many exploit code-level vulnerabilities in the application code [2]. Major software-development companies, including SAP, embed in their development process activities (e.g., dynamic and static security testing [3]) to identify vulnerabilities early during the development of their software system. Nevertheless, their security development lifecycle (see, e.g., [4] for Microsoft's security development lifecycle) includes also a process for addressing vulnerabilities identified after the software is released.

Analyzing and fixing security vulnerabilities is a costly undertaking. Surely it impacts a software's time to market and increases its overall development and maintenance cost. But by how much? To answer this question directly, one would need to trace all the effort of the different actions that the developers undertake to address a security issue: initial triage, communication, implementation, verification, porting, deployment and validation of a fix. Unfortunately, such a *direct* accountability of the individual efforts associated with these action items is impossible to achieve, last but not least due to legal constraints that forbid any monitoring of the workforce. One must therefore opt for *indirect* means to relate quantitative, measurable data, such as the vulnerability type, the channel through which it was reported, or the component in which it resides, to soft human factors that correlate with the time it takes to fix the related vulnerabilities. But, which factors impact this fixing effort positively or negatively?

This paper aims to identify the factors that impact the vulnerability fix time in SAP software. (We use vulnerability fix time and vulnerability fix effort interchangeably.) For this work we interviewed 12 experts who contribute to addressing security vulnerabilities at SAP, one of the largest software vendors worldwide, and the largest in Germany. The study comprises teams located in different countries, developing diversified products. The work led to the discovery of 65 factors impacting the vulnerabilities fix time, which we classified into 8 categories. The factors could be used to estimate the required effort to fix vulnerabilities and to improve the secure development activities.

This paper is organized as follows. First, we give an overview of related work (Section 2) and discuss secure software development at SAP (Section 3). Next, we describe the research approach that we use in this work (Section 4), report about our findings (Section 5) and discuss the impact and the limitations of the study (Section 6). Subsequently, we discuss some of the lessons we learned from the study (Section 7) and conclude in Section 8.

2 Related work

Several pieces of research investigate the time it takes to fix software defects [5,6]. For instance, Hewett and Kijsanayothin applied machine-learning algorithms to defect data collected from the development of a large medical-record system to predict the duration between the time of identification of the defect and the validation of the appropriate fix [6].[3] Opposed to this previous work, we (1) focus on security vulnerabilities, not functionality errors, and (2) include in our model "human factors" such as organizational issues that cannot directly be derived from automatically collected data. In this work, as a first step, we determine the relevant factors.

[3] Among other things, the duration includes the time the defect is in the repair queue after being assigned to a developer.

Table 1: Examples of time required for fixing vulnerabilities [7].

Vulnerability type	**Average fix time (min)**
Dead Code (unused methods)	2.6
Lack of authorization check	6.9
Unsafe threading	8.5
XSS (stored)	9.6
SQL injection	97.5

Software defects have been found to be correlated with software complexity [8], which is measured, e.g., using the size of the code and the density of its control instructions. There is a general hypothesis that software complexity is also correlated with the existence of vulnerabilities e.g., [2]. This hypothesis is often false. For example, Shin et al. [9] and Chowdhury et al. [10] found that the complexity metrics of open-source software such as Firefox only weakly correlate with the existence of vulnerabilities in those systems. Thus, the factors (e.g., code complexity) that apply to software-defects based models do not necessarily apply to vulnerabilities based models.

The only work we know that evaluates vulnerability fix time was performed by Cornell, who measured the time the developers spent fixing security vulnerabilities in 14 applications [7]. Table 1 shows the average time the developers take to fix vulnerabilities for several vulnerability types. The measured time comprises only the fix-execution phase, which includes the environment setup, implementation, validation, and deployment of the fix. Cornell found that the percentage of this time spent on the implementation of the fix is only between 29% and 37% of the time spent in the execution phase. The author was unable to measure the time spent on the inception (including risk assessment) and planning phases because the collected data were too inconclusive. Cornell found also that there are vulnerability types that are easy to fix, such as dead code, vulnerability types that require applying prepared solutions, such as lack of authorization, and vulnerability types that, although simple conceptually, may require a long time to fix for complex cases, such as SQL injection.

The vulnerability type is thus one of the factors that indicate the vulnerability fix time but is certainly not the only one. This paper aims to identify as many factors as possible that will likely impact the vulnerability fix time, factors that could be collected automatically but also factors that can only be inferred indirectly by observing how human analysts and developers go about fixing vulnerabilities.

3 Secure Software Development at SAP

SAP has a very diverse product portfolio: for example, a SAP product might be a small mobile application or an enterprise resource planning (ERP) system. Similarly, a large number of different programming languages and frameworks

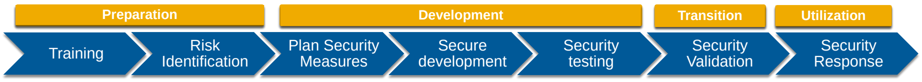

Fig. 1: High-level Overview of the SAP Security Development Lifecycle (S^{2}DL)

are used during their development and many different environments (e.g., web browsers, operating systems) are supported. Moreover, SAP develops also frameworks, such as SAP Netweaver, that are both offered to customers and used to build other SAP products. Finally, SAP product portfolio ranges from on-premise products to cloud offerings (including private clouds, public clouds, and hybrid clouds).

To ensure a secure software development, SAP follows the SAP Security Development Lifecycle (S^{2}DL). Figure 1 illustrates the main steps in this process which is split into four phases: preparation, development, transition, and utilization. For our work, the second half of the S^{2}DL is important:

- during the actual software development (in the steps *secure development* and *security testing*) vulnerabilities are detected, e.g., by using static and dynamic application security testing tools [11,3];
- *security validation* is an independent quality control that acts as “first customer” during the transition from software development to release , i.e., security validation finds vulnerabilities after the code freeze, (called correction close) and the actual release;
- *security response* handles vulnerabilities reported after the release of the product, e.g., by external security researchers or customers.

To allow the necessary flexibility to adapt this process to the various application types developed by SAP as well as the different software development styles and cultural differences in a worldwide distributed organisation, SAP follows a two-staged security expert model:

1. a central security team defines the security global processes (such as the S^{2}DL, provides security trainings, risk identification methods, offers security testing tools, or defines and implements the security response process;
2. local security experts in each development area/team are supporting the developers, architects, and product owners in implementing the S^{2}DL and its supporting processes.

If a vulnerability is detected, developers and their local security experts follow a four step process: (1) analyze the vulnerability, (2) design or select a recommended solution, (3) implement and test a fix, and (4) validate and release this fix. In the security testing process, a security expert is expected to inspect the analysis results of any utilized testing tool and determine for each of the reported findings whether it is exploitable, and consequently requires fixing. The vulnerability then gets assigned to a developer who implements the suggested solution. The fix is verified by a retest of the code with the same testing rules. The fix is considered to be successful when the test passes.

While this process is the same, regardless if the vulnerability is in released code or current development code, certain administrative steps exist prior to the first step but the steps necessary to release a fix and the involved parties differ. For vulnerabilities in not yet released code, the process is locally defined by the development team and, usually, very lightweight. For vulnerabilities in *released* software, the security response team, developers and security experts are mainly involved in the first three fixing phases and the maintenance team (called IMS) is mainly involved in the last phase. Fixes of released code are reviewed and validated by the central security team. These fixes are shipped in security notes or support packages for customers to download. Security notes are patches included in support packages. Support packages are functional updates that also contain the latest security notes.

4 Research approach

We conducted a qualitative case study to identify the factors that impact the vulnerability fix time for vulnerabilities reported to or within SAP. A case study is an empirical inquiry that investigates a phenomenon in its real-life context [12]. This study uses expert interviews as data source; that is, interview of security experts, coordinators, and developers who contribute to fixing vulnerabilities. The aim of the interviews is to use the experiences of the interviewees to identify the factors that impact the time they spend in contributing to fixing vulnerabilities.

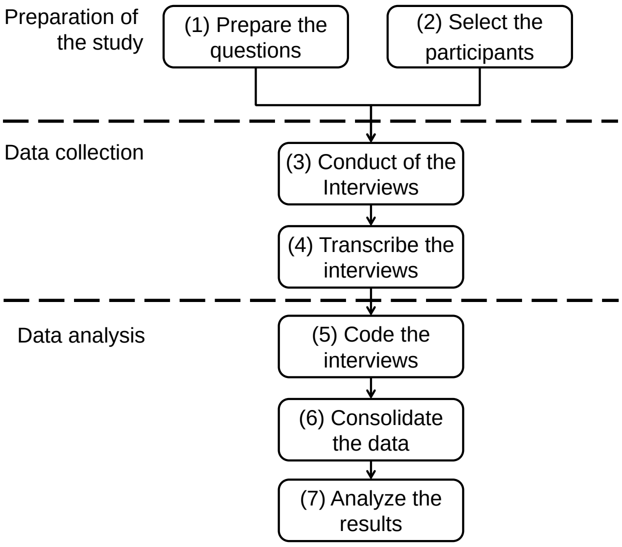

Fig. 2: The steps of the case study.

Figure 2 depicts the study process. It has three phases: study preparation, data collection, and data analysis. The description of the three phases follows.

Preparation of the study. Initially we reviewed a set of documents that describe the processes of fixing vulnerabilities and discussed these processes with SAP security experts. We used the knowledge to develop a set of interview questions. Then, we met three security experts in pre-interviews to learn the fixing process further and to check the questions that we developed. We summarized the questions (Step 1 of Figure 2) in an interview protocol. An interview protocol describes the procedural and the main questions that should be used in the interviews [13]. We choose semi-structured questions, which allowed us to capture similar type of data (e.g., roles, pinpoints, and recommendations) across all the interviews while being flexible to explore reported content.

Fixing vulnerabilities at SAP requires collaboration of people having different roles, who could be located in different cities and countries. We considered this contextual factor and invited representatives (Step 2 of Figure 2) of the different roles located in several offices to participate in the study. Twelve participants accepted: nine were from Germany and three were from India. The participants were NetWeaver experts, application-layer experts, and experts in developing customer specific applications. Their roles were security experts, developers, coordinators, and project leads. This method of selecting participants complies with the maximum-variation-sampling approach [14]–a preferred participants sampling method.

We scheduled one hour for each interview. We sent the participants the interview protocol so they could prepare for the interview; e.g., prepare examples.

Table 2: Interview questions.

Question	**Targeted information**
1. What is your role in fixing vulnerabilities?	The role of the participant
2. How do you get the information about security vulnerabilities? And what do you do with the information?	The source of information about vulnerabilities, the steps performed by the interviewee in fixing vulnerabilities, and the tools used by the interviewee to fix the vulnerabilities
3. What are the worst and best cases of vulnerabilities you worked on? And how did do you address them?	The factors that impact the vulnerability fix time
4. How much time did you spend on [..]? Why did [it] take that long?	The factors that impact the vulnerability-fix time
:q5. How would we improve the way you work?	The factors that impact the vulnerability-fix time

Data collection. We conducted the 11 interviews (Step 3 of Figure 2) within one week (One of the interviews was conducted for 2 hours with 2 interviewees

Table 3: Used coding schema.

Code class	Description
Meta	The role of the interviewee and their experience with SAP products and with fixing vulnerabilities
Used tools	The tools used in fixing the security vulnerabilities.
Process participants	The roles and teams that the interviewee collaborated with in fixing the vulnerabilities they worked on.
Process activities	The activities that the interviewee performs when fixing security vulnerabilities, which are classified into pre-analysis or risk assessment, analysis, design, implementation, test, release activities.
Information for activities	The information used for analysis including risk assessment analysis, design, implementation, test, and release activities.
Factors	The factors affecting vulnerability fix time for the case of generic solutions (a generic solution is a way to address all the instances of a specific vulnerability type, e.g., XML code injection) and also the case of specific solutions.
Complementary information	This includes generic comments, comments related to vulnerability fix time, pain points (issues), and improvement recommendations.

as they requested.) and initially used the questions that we prepared in the interview protocol. We let the interviewee lead and we probe issues in depth, when needed, using questions such as "Could you provide an example?" Nevertheless, we realized shortly that it was difficult for the interviewee to provide us with the maximum information related to our research goal. We adapted the questions of the interview protocol to the ones provided in Table 2. The adaptation is discussed in Section 7. Also, some interviewees provided us with tool demonstrations since they were aware about the interview protocol.

Next, we transcribed the interviews (Step 4 of Figure 2) using the tool F4.[4]

Data analysis. Subsequently, we proceeded to coding the interviews (Step 5 of Figure 2); that is, identifying from each transcript the codes, i.e., themes and abstract concepts (e.g., code the text "I have been fixing these issues for 5 years" as "experience in fixing vulnerabilities").[5] In this step, two of the authors coded successively 3 sample interviews using the Atlas.ti tool,[6] discussed the code patterns they found, and agreed on a coding schema for the study, which is shown in Table 3. (The coding schema allows grouping the codes extracted from the interview in classes that together answer the main research question [14].) Both researchers coded each of the 11 interviews using the selected coding schema and merged their reports in summary reports.[7] Then, we sent to each interviewee

[4] `https://www.audiotranskription.de/english/f4.htm`

[5] A code is a short phrase that assigns a summative, essence-capturing, and/or evocative attribute for a portion of text [15].

[6] `http://atlasti.com/`

[7] The merge involves also discussing coding mismatches related to the difference in understanding the interviewee.

the summary report of their interview and asked them to verify the report and answer some clarification questions. The validation helps in obtaining objective results but was also important to allow the interviewee to remove any information they did not want to be processed. We ensured the anonymity of the interviewees to promote free and open discussions.

Afterwards, we merged the codes of the verified coded reports (Step 6 of Figure 2) considering the semantic similarities between the codes extracted from different transcripts. In addition, we computed the frequency of each code in the class "Factors," that is, the number of interviewees mentioning the code. We were reluctant to generalize the factors because we did not want to bias the results with the researcher' opinions.

Thereafter, we presented the findings to the experts and the interviewees in a public meeting (Step 7 of Figure 2). We used the frequencies of the codes as indicators (but not assertive) of the factors' importance.[8]

5 Study results

This section presents the results of the interviews. It discusses the vulnerability-fixing process identified at SAP and the factors that impact vulnerability-fixing time along with their classification.

5.1 Vulnerability-fixing process

Each interviewee described a set of activities that they perform to fix vulnerabilities. The activities described by the different interviewees were sometimes incoherent–Section 7 discusses the challenges. However, in many ways the interviewees follow a high-level vulnerability-fixing process, which is depicted by Figure 3. The process starts when a security expert gets notified about vulnerabilities, e.g., from customers and researchers, or when a developer identifies a vulnerability using, e.g., a code-analysis tool. The vulnerability is initially pre-analyzed, e.g., to assess its exploitability, its risk and the availability of knowledge and information to fix it. This results in three cases.

Case 1. If the type of the vulnerability is known and documented, the developer proceeds to analyze the code related to the vulnerability, to design and implement a solution, and then to test it—using the technique that was used to identify it.

Case 2. If the vulnerability type is known and documented by the central security team but the development teams (e.g., cloud applications, mobile applications, etc.) did not encounter such vulnerability before, this team collaborates with the central security team to analyze the identified vulnerability and design a solution that applies to the product area as such.

[8] Recall that data extracted from interviews could not be used to derive statistical assurance of the conclusions since the collected information is descriptive.

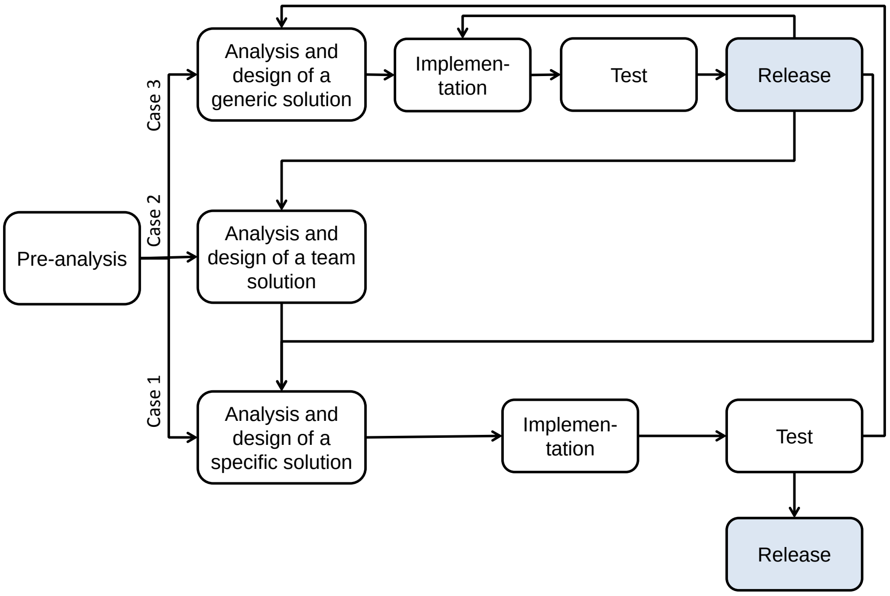
Analysis and design of a generic solution
Implemen-tation
Test
Release
Case 3
Case 2
Pre-analysis
Analysis and design of a team solution
Case 1
Analysis and design of a specific solution
Implemen-tation
Test
Release

Fig. 3: High-level vulnerability-fixing process (simplified).

Case 3. If the vulnerability type is not known before, the central security team collaborates with the experts and developers from the different areas to develop a generic solution for the vulnerability. A *generic solution* considers the different product areas, the different used technologies, and the different applicable programming languages. In addition, the security experts collaborate with the framework experts to implement libraries that the developers can use to avoid the vulnerability in the future, e.g., by using data-validation methods to avoid SQL injections; and also develop guidelines that the developers can use to address vulnerabilities of such type.

5.2 Factors that impact the vulnerability-fix time

We identified 65 factors that impact the vulnerability-fix time, each was reported by at least one interviewee. We categorized the factors based on common themes, and those that did not belong to these themes into the category "other." These categories may be generalized or consolidated further, however, we expect that such activity may be influenced by the researchers' opinions. Table 4 lists the categories, along with the number of factors that belong to each category and the number of interviewees who mentioned one or many of these factors. Table 6 of Appendix A provides the complete list of the factors that we identified.

Table 4: Classification of the factors that impact the vulnerability fix time.

Factor Categories	**Number of factors**	**Frequency**
Vulnerabilities characteristics	6	9
Software structure	19	10
Technology diversification	3	5
Communication and collaboration	7	8
Availability and quality of information and documentation	9	9
Experience and knowledge	12	11
Code analysis tool	4	4
Other	4	4

We next discuss the categories in details. To preserve the anonymity of the interviewees we identify them using identifiers, without descriptive information.

Vulnerabilities characteristics. This category includes 6 factors that are exclusively related to the type of vulnerability. These factors are reported in 9 (about 82% of the) interviews. For example, P01 believes that vulnerability types do not indicate the fixing time but later in the interview they find that "code injection" vulnerabilities are difficult to fix. Thus, vulnerability characteristics are commonly considered when discussing vulnerability fix time (e.g.,in [7]) and our results enforce the position.

Software structure. This category includes 19 factors that are related to the code associated with the given vulnerability. These factors are reported in 10 (about 91% of the) interviews. For example, P02 finds that "if the function module is the same in all these 12 or 20 releases then [..] I just have to do one correction." Generally, the interviewees find that software structure impacts the easiness to address vulnerabilities. This can also be observed from Table 1, where SQL injection vulnerability takes the most time to fix while it is conceptually easy to fix. The reason is that the complexity of the code that generates the query makes it difficult to identify the cause of the vulnerability and to fix the issue while not breaking the functional tests.

Technology diversification. This category includes 3 factors that are related to the technologies and libraries supported by the components associated with the given vulnerability. These factors are reported in 5 (about 45% of the) interviews. For example, P03 had to develop several solutions for a vulnerability related to random-number generation, "one for Java, one for ABAP, one for C, C++." Thus, since SAP products support different browsers, languages, and use diverse libraries, such as XML parsers, vulnerability fixes need to support these technologies as well, increasing the overall time and effort required.

Communication and collaboration. This category includes 7 factors that are related to the communication and collaboration in fixing vulnerabilities. These factors are reported in 8 (about 73% of the) interviews. For example, P04 finds that “even for one team there are multiple locations and multiple responsibilities and the team at [..] is not aware” and finds that “the local teams are very smooth.” Developing software at SAP involves teams located in different locations. Thus, the smoothness of the communication and collaboration between the stakeholders impacts the time spent to fix the vulnerabilities.

Availability and quality of information and documentation. This category includes 9 factors that are related to the availability and the quality of information (e.g., contact information of the security experts, and uses of components) and guidelines to address vulnerabilities. These factors are reported in 9 (in 82% of the) interviews. For example, P05 claims that a lot of time is spent on collecting information. They state:“it was taking a long time because we need to find out what are the different frameworks, what are the different applications [..] and once we had this information, we were able to use it.”

Experience and knowledge. This category includes 12 factors that are related to the experience and knowledge about the given vulnerability and the related code. It is reported in all the interviews. For example, P01 finds that “colleagues who have some background in security are able to fix them faster than the developer who is fixing the security issues for maybe the first or second time.” This category of factors is often ignored in existing studies because those studies rely on data archives which do not include such human factors.

Code analysis tool. This category includes 4 factors that are related to the use of code analysis tools. This category of factors is reported in 4 (36% of the) interviews. P13 for example says “what you are doing is to find out if the tool was not able to find out where is the source, where is the data coming from so find out if there is external input in this where clause or in parts of external inputs.” This implies that developers spend less time to fix a given vulnerability if the tool is accurate in providing the information related to the causes of the vulnerabilities. This category of factors is often neglected because it is not common that organizations use several code-analysis tools, so their impact on the fixing time cannot be compared.

Other. This category includes 4 factors that we were not able to classify in the above 7 categories and which do not belong to a common theme. These factors are reported in 4 (36% of the) interviews.

We note that the number of factors that belong to a given category does not assert the importance of this category but could be informative. The reason is that the descriptive nature of the interviews does not support such assertion.

5.3 Discussion

When developing the factor categories, we did not differentiate the factors based on whether the given vulnerability is generic or specific. It is true that generic vulnerabilities, i.e., vulnerabilities that are identified for the first time but apply to several products, are often addressed with generic solutions designed by the central security team. Vulnerability analysis, solution design, documentations and the provisioning of guidelines are time consuming and addressing such vulnerabilities may take years. In contrast, specific vulnerabilities, i.e., vulnerabilities that are known and apply to one product, are mostly addressed by developers in collaboration with the development team's security expert. The reason for not considering this is that most of the factors apply to both generic and specific vulnerabilities—with some exceptions.

As a second note, the positive or negative influence of each of the identified 65 factors that impact the vulnerability-fix time were not identified in the interviews and may depend, e.g., on the product. In fact, the developers gave contradicting perceptions on the influence of some factors, such as the factor risk exposure level. Also, the number of interviewees who mentioned each of the factors does not indicate the influence of the factor. We will evaluate the concrete influence of each factor in the next stage of the project.

6 Impacts and limitations of the study

This section discusses the impacts of the study w.r.t the state of the art and the study validity; that is, to what extent the results are objective and sound [16].

6.1 Impacts of the study

Previous work used the attributes of collected data as factors for analyzing facts about vulnerabilities. The results of this work show that in practice there are numerous factors that impact the vulnerability fix time that should be considered. A comprehensive model for predicting the vulnerability-fix time should consider these potential factors and not only rely on the ones that are readily available in mass-collected data.

The identified vulnerabilities fixing process (Figure 3) and the 8 factor categories may be "expected," especially from a big company like SAP. The study confirms this expectation; it makes these expectations facts that could be used for further work. The 8 factor categories indicate areas for improvement to reduce the vulnerability fix time. For example, the software structure (e.g., the dynamic construction of code and data, the cross-stack interdependency) and the use of external technologies, could be partly monitored and controlled to predict and/or reduce the vulnerability fix time, and thus, the cost of fixing vulnerabilities. In addition, experience and knowledge can be addressed with specific trainings; effectiveness of code analysis tools could be improved by enhancing the vulnerability checks, in particular their precision, and by enhancing the tool' functionalities; issues related to the availability and quality of information and documentation can also be improved by using easily accessible documentation.

In addition, we found in this study that the developers cannot identify the vulnerability fixing factors by themselves easily, except e.g., vulnerability type, which we discuss in Section 7. Though, they recognize the factors if extracted from their interviews. Thus, the results are not "explicit" knowledge to developers. This paper makes the information common knowledge.

6.2 Limitations of the study

We discuss now the limitations of the study according to the commonly used validity aspects [16].

Construct validity. We took several measures to ensure a valid relation between the performed study and the goal of the study. First, we performed three interview tests to test the interview questions. In addition, we adjusted the interview questions after the initial interviews to be more efficient in getting information. Second, we collected the information from twelve interviewees, who are located in different cities/countries and have different roles. Third, we avoided to use the researcher' opinions in working with the data, e.g., we avoided generalizing the factors extracted from the interviews.

The study has two limitations w.r.t. construct validity. First, we provided the participants with the main interview questions so they could prepare for their interviews. Thus, some participants may have prepared replies that may influence the study results. (We believe that the advantages of the measure are higher than the risk it created to the study.) Second, we used only one method to collect the data, that is, interview domain experts. Other methods that could be used to cross-validate the results include to use of data collected from the development process. Nevertheless, we observed that the attributes of collected data are among the identified factors.

Internal validity. We took two measures to ensure a causal relationship between the study and the results of the analysis. First, we tell the interviewees at the opening of the interviews that the goal is to identify the factors that impact the vulnerability fix time.[9] Second, we did not offer any compensation to the participants, which, if done, may affect the results.

The study has two limitations w.r.t. internal validity. First, we were able to only interview two developers who currently fix vulnerabilities. The other participants have other roles in fixing vulnerabilities but most of them have developed vulnerability fixes previously.[10] Second, we did not take measures to prevent the participants from imitating each others in the response, though we believe that the participants did not talk to each other about the interviews.

Conclusion validity. We took several measures to ensure the ability to draw correct conclusions about the relationship between the study and the results of the analysis. First, we sent each interviewee a short report about the data we extracted from the interview we conducted with them to ensure that we have a common understanding; that is, we performed member checking [17]. Second, two researchers coded each interview and we merged the collected data [14]. The measure should reduce the subjectivity of the results.

External validity. This validity concerns the conditions to the generalization of the results. The study was conducted in the same company and was related to one vulnerability fixing process. However, given the diversity of the products

[9] this mitigates the threat ambiguity of the direction of the causality relationship.

[10] The limitation is related to the selection of participants.

Table 5: Summary of the interview protocol.

Opening		Thank the interviewee for accepting to participate and request for permission to record the interview.
Questions.	1.	What are the steps that, in general, you follow to fix security vulnerabilities?
	2.	What is the distribution of your time among planning, meeting, and implementing vulnerability fixes for the last week? Did you have any other major activity in fixing vulnerabilities?
	3.	What are the major characteristics of complex vulnerabilities that you fixed last week? Are there other challenges that make simple vulnerabilities time consuming?
	4.	What are the factors that quantify these characteristics?
	5.	How can we improve and ease the fixing process?
Closing.		Thank the interviewee for sharing his/her experience and knowledge and inform him/her about the next steps for the study.

(and their respective domains, e.g., mobile, cloud.) being developed at SAP and the diversity of the developers' cultures and countries of residences, we believe that the results of the study could be generalized, especially within SAP, without high risk.

7 Lessons learned

This section describes the lessons we learned from the case study with respect to formulating interview questions, conducting interviews, and analyzing software development processes of a big software organization.

Interview protocol and questions. We produced an interview protocol, summarized in Table 5. The first interviews showed that the interviewees had, in general, difficulties in answering questions 3 and 4, initially developed as main questions to achieve the study goal. This was due to the "what" type of asked questions (i.e., "what are the factors?") that require enumeration of elements while we should be limited to "how" and "why" type of questions.[11] To enhance the communication we transformed the questions accordingly (see question 3 and 4 of Table 2) and encouraged the participants to tell us *their own stories* [13] about complicated and easy vulnerabilities they addressed and the challenges they faced. Thus, with indirect questions we derive the factors that impact the fix development time from the reasons that make fixing a given vulnerability complicated, the challenges that the interviewees faced herein, and their improvement recommendations (question 5 in both tables).

Interview conduct. We learned that some participants in interviews conducted in organizations mainly participate to deliver a message or impact the study

[11] "What" type of questions are easy to answer when the purpose is to describe a concept/object.

results. We learned to encourage the participants to talk freely for some time to build up a trust relationship, since the information they provide when getting into a flow may be important. The risk herein was the limited interview time and thereby the challenge to change the discussion smoothly such that they answer the interview questions and do not use the interview to talk about subjects not related to the research goal.

Analysis of the software development processes. The intuitive approach to identify the vulnerability fix time is to define the fixing process and its phases. Following this approach, our initial attempt was to identify the different roles of the participants in the process, the activities performed in the phases and the information created and consumed by each role. We derived inconsistent models with a big variety of cases. This is due to the different perspectives of the interviewees; they work with different programming languages and tooling, have different (and multiple) roles, have expertise in different product areas, and are members of different teams, and use their own internal social network to simplify the work. In addition, there were process improvements and each of the interviewees reported about the process versions that they worked with. The open structure of the interviews and the participation of long time employees made it difficult to identify a consistent fixing process. Therefore, we focused on the identification of the factors independent of the phases.

8 Conclusions

This paper reports about a case study we conducted at SAP SE to identify the factors that impact the vulnerability fix time. The study found that, for big development organizations such as SAP, there are numerous factors that impact the vulnerability fix time. We identified 65 factors, which we grouped into 8 categories: vulnerabilities characteristics, software structure, diversity of the used technology, communication and collaboration smoothness, availability and quality of the information and documentation, expertise and knowledge of developers, efficiency of the static analysis tool, and other.

The study was conducted at one organization, SAP SE, which may limit the generalization of the results. We believe that the limitation is weak because SAP development groups simulate different organizations, each has independence and specificities such as location, used programming language, and products area.

The common approach in investigating vulnerability fix time (and other facts related to vulnerabilities) is to apply machine-learning techniques on historical data related to open-source software. This work shows the limitation of this approach since it is constrained by a limited number of data attributes while the factors that potentially influence these facts are numerous.

The results of this work are being used to improve the vulnerability fixing process and to develop a model for predicting the cost of fixing vulnerabilities.

Acknowledgments

This work was supported by SAP SE, the BMBF within EC SPRIDE, and a Fraunhofer Attract grant. The authors thank the participants in the study.

References

1. Katzeff, P.: Hacking epidemic spurs security software stocks. http://news.investors.com/investing-mutual-funds/021915-740082-revenues-are-up-for-security-software-firms.htm (Feb. 2015) Investor's business daily of 02/19/2015.
2. McGraw, G.: Software Security: Building Security In. Addison-Wesley Software Security Series. Pearson Education Inc, Boston, MA, USA (2006)
3. Bachmann, R., Brucker, A.D.: Developing secure software: A holistic approach to security testing. Datenschutz und Datensicherheit (DuD) **38**(4) (apr 2014) 257–261
4. Howard, M., Lipner, S.: The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software. Microsoft Press (2006)
5. Hamill, M., Goseva-Popstojanova, K.: Software faults fixing effort: Analysis and prediction. Technical Report 20150001332, NASA Goddard Space Flight Center, Greenbelt, MD United States (Jan. 2014)
6. Hewett, R., Kijsanayothin, P.: On modeling software defect repair time. Empirical Software Engineering **14**(2) (2009) 165–186
7. Cornell, D.: Remediation statistics: What does fixing application vulnerabilities cost? In: RSAConference, San Fransisco, CA, USA (Feb. 2012)
8. Khoshgoftaar, T.M., Allen, E.B., Kalaichelvan, K.S., Goel, N.: Early quality prediction: A case study in telecommunications. IEEE Softw. **13**(1) (January 1996) 65–71
9. Shin, Y., Williams, L.: Is complexity really the enemy of software security? In: Proc. of the 4th ACM Workshop on Quality of Protection. QoP '08, Alexandria, VA, USA (Oct. 2008) 47–50
10. Chowdhury, I., Zulkernine, M.: Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. Journal of Systems Architecture **57**(3) (2011) 294 – 313 Special Issue on Security and Dependability Assurance of Software Architectures.
11. Brucker, A.D., Sodan, U.: Deploying static application security testing on a large scale. In: GI Sicherheit 2014. Volume 228 of Lecture Notes in Informatics. (mar 2014) 91–101
12. Yin, R.K.: Case Study Research: Design and Methods. Sage Publications, Beverly Hills, CA (1984)
13. Jacob, S.A., Furgerson, S.P.: Writing interview protocols and conducting interviews: Tips for students new to the field of qualitative research. The Qualitative Report **17**(42) (Oct. 2012) Art. 6, 1–10
14. Brikci, N., Green, J.: A guide to using qualitative research methodology. http://www.alnap.org/resource/13024 (Feb. 2007)
15. Saldana, J.: The Coding Manual for Qualitative Researchers. SAGE Publications Ltd, London, UK (2009)
16. Wohlin, C., Runeson, P., Host, M., Ohlsson, M., Regnell, B., Wesslen, A.: Experimentation in Software Engineering. Springer-Verlag, Berlin Heidelberg (2012)
17. Seaman, C.: Qualitative methods in empirical studies of software engineering. IEEE Transactions on Software Engineering **25**(4) (1999) 557–572

Appendix A: Factors that impact the vulnerability fix time

Table 6: Factors that impact the vulnerability fix time. (The column "Freq." indicates the number of interviews–out of 11–where the given factor is mentioned.)

Factors	**Freq.**
Vulnerabilities characteristics	
Vulnerability type	5
Risk exposure level (CVSS)	3
Priority of the vulnerability fix	2
Availability of a generic solution	2
Simplicity of the solution type, e.g., exception catching	2
Source of the vulnerability, e.g., input and remote function call	1
Software structure	
Number of affected applications and framework releases	7
Need of the solution to change the framework, e.g., as an API	5
Number of software layers that should be analyzed	5
Number of the code changes and the distribution of their locations	4
Dependency on external software and/or technology	3
Code related to the vulnerability includes generated code	3
Complexity of the related code and reliance on configurations	2
Level of code smells (e.g., high, low)	2
The issue is related to the dynamic generation of queries	2
Dynamic generation of the input data that cause the vulnerability	2
The code related to the vulnerability involves dynamic function calls	2
Existence of the issue in shipped products	1
Need for customer intervention to apply the fix	1
Need to divide the fix in several support packages	1
Similarity of code related to the issue in the releases of the software	1
Code related to the vulnerability is in new development	1
Components related to the issue	1
Length of the data field related to the issue (for SQL injection)	1
The vulnerability is in an unused code	1
Technology diversification	
Diversity and complexity of supported technologies, e.g., browsers	5
Number of supported programming languages	2
Number of customers that have specific functionalities that should be preserved but may be affected by the vulnerability fix	1
Communication and collaboration	
Number of people involved in the fixing process	5
Dispersion of the locations of the communicating parties	4
Availability of the responsible developer	1
The software is developed by an external developer	1

Continued on next page

Table 6 – *Continued from previous page*

.**Factors**	**Freq.**
Existence of concurrent changes to code of dependent components	1
Easiness to convince the developers to apply vulnerability fixes	1
The code related to the issue is owned by other teams	1
Availability and quality of information and documentation	
Availability of information about the developer responsible for the fix	5
Availability and usability of the guidelines explaining the vulnerability	5
Availability of information about contact persons for support	2
Availability of information about the framework changes that cause the vulnerability in the applications	1
Quality of the documentation for technology types (e.g., Hana)	1
Availability of information about the work progress on the issue	1
Availability of information about contacts in other development areas	1
Difficulty to get the information about affected software	1
Difficulty to identify the use of affected code by other components	1
Experience and knowledge	
Developer expertise with fixing security vulnerabilities	7
Developer knowledge and experience about the code	7
Experience with SAP software, processes and tools	2
Developer knowledge about the vulnerability	2
Security expert knowledge of the language and affected technologies	1
Experience with the application technology	1
Knowledge about the software architecture	1
Knowledge about the development setup of affected software	1
Knowledge about the development processes for NetWeaver products	1
Availability of security expert in the area responsible for the issue	1
The coding and testing strategy of developers	1
Experience in the impacts of changes on the related products	1
Code analysis tool	
The issue is among a set of issues of the same vulnerability type and in the same code component	2
Lack of motivation due to high rate of false positives	2
The issue is for projects addressing results identified by external tools	1
The failure of the security checks to identify true positives	1
Other	
Commitment of the stakeholders	1
Source of the notification about the vulnerability, e.g., customer	1
Number of externally reported incidents	1
The issue is a false positive that does not require fixing	1